USENIX Association

Proceedings of the 14th Systems Administration Conference (LISA 2000)

New Orleans, Louisiana, USA
December 3–8, 2000

USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

FTP Mirror Tracker: A Few Steps towards URN

Alexei Novikov – Institute of Theoretical and Experimental Physics, Moscow, Russia
Martin Hamilton – Loughborough University, UK

ABSTRACT

FTP Mirror Tracker[1] is a software package (written in Perl and C++) that enables transparent, user-controlled redirection to the nearest anonymous FTP mirror sites that are exact replicas of the original source. This redirection can be achieved by using a Web Cache server or by making HTTP requests to the FTP Mirror Tracker directly. The Mirror Tracker also has internal URN support and can be used as a URN resolver for FTP requests. Underlying the system is a MySQL database recording FTP mirror site details. In this report we explain how this database is constructed, and show how it may be used – directly by end users, and under the policy based control of Web Cache and mirror service administrators.

Introduction

Although FTP traffic passing through the modern Internet only accounts for a small fraction of request transactions, its bandwidth utilization is significant. For example, FTP accounted for between 7% and 11% of the incoming traffic on the JANET[2] network's links to the United States for every month in 1999.

There is a long standing Internet convention that sites which are particularly large (e.g., operating system distributions) or popular (e.g., the Starr Report) will be widely replicated – usually by volunteer effort. The replication process, which typically takes place on a daily basis, is usually referred to as mirroring. Mirroring software exists for replicating Web (HTTP), FTP and (by prior arrangement) arbitrary content, e.g., GNU wget [2], mirror [3] and rsync [4]. In recent years some formalization of this role has taken place, e.g., with the establishment of the UK Mirror Service [5] for JANET users, and the AARNet2 Mirror Archive [6] for Australian academic and research users. Localizing what might well be international (and chargeable) traffic to geographically and/or topologically nearby mirror sites is a challenging task.

A number of approaches to brute-force indexing of the FTP namespace have been attempted, notably the Archie [7] system from Bunyip, CNET's Shareware.com [8] and Lycos/FAST FTP Search [9]. The user interfaces for searching these systems typically allow the end user to supply full or partial filename details or regular expressions to match, and return a list of the URLs where files matching the search criteria can be found. Because of the difficulty in knowing where on the Internet is the best topological/policy match for a client, only minimal attempts have been made to provide tailored output on a per-user basis.

[1]http://squid.itep.ru/ mirrored at http://wwwcache.ja.net/mirrors/MirrorTracker/

[2]JANET [1] is the UK's Higher Education and Research Network

The FTP Mirror Tracker Design in Brief

The core of the FTP Mirror Tracker software is a robot which traverses through a list of anonymous FTP servers which it has been told to visit, connects to each of them in turn, fetches their public directory tree content (using the FTP `ls -lR` command), analyses it and creates a unique identifier (actually an MD5 [10] digest value) for the content of each directory and the tree of the directories (using a summarizer program). These identifiers and the FTP URL paths they refer to are stored in a MySQL database [11] using the Perl [12] DBI database interface [13]. In turn, the MySQL database is accessed by a variety of programs which collectively form the "user interface" to the system.

The processed data created by the Mirror Tracker summariser is also made available for use by other services, e.g., for sharing with other Mirror Tracker servers. By default each Mirror Tracker maintains copies not only of the data for the Internet domains it is responsible for, but also the data for the domains which are indexed by other Mirror Trackers. A "root" Mirror Tracker has been established in Moscow, Russia to help bootstrap this process.

From a user standpoint, the basic operation of the FTP Mirror Tracker is to take a URL and return a list of the alternative URLs where this same material may be found – subject to search criteria such as the top level domain(s) required in the search results. This is done by finding the unique identifier associated with the requested URLs, and checking to see whether any other entries in the table(s) in question have the same unique identifier.

We will use the phrase "unique identifier" rather than "MD5 digest value" throughout this paper, since there is no actual requirement that MD5 digests be used as the resource collection identifier. The uniqueness and location independence of our identifiers also makes them attractive as a global naming system, though it should be noted that the way they are calculated means that they are not persistent over time. This means that they are not truly suitable for use as URNs. However, since there is very little practical deployment experience with URNs, we have chosen to ignore this problem for the time being.

The Gory Details

The FTP Mirror Tracker architecture consists of the following components:

A robot which collects directory listing data from the FTP servers it has been configured to track.
- A summarizer which creates MD5 digests of the directory listings.
- A digest exchanger, for sharing the digests with other servers.
- A back end database – currently MySQL.
- Various frontend programs.

We will describe in short the design of each of these components, and how they have been implemented.

The Mirror Tracker robot

The function of this component is to gather raw directory listings from FTP servers for processing by the summarizer. The robot has been implemented as two Perl programs – robot, which parses the list of the FTP servers for each domain begin indexed and forks a second Perl script, ftp_list, to actually fetch the directory listing for the server.

If it was able to start an anonymous FTP session with the target server, ftp_list proceeds to check whether there is an ls-lR.gz file in the root or /pub directories on the server. If one that it is reasonably fresh (and not empty) is present, this is fetched, otherwise the ls -lR command is issued from the top level directory of the anonymous FTP server in order to produce a recursive directory listing.

The summarizer

This is the heart of the FTP Mirror Tracker. Its job is to parse the directory listings of the FTP servers which are being tracked (fetched by the robot), analyze them, and create MD5 digests based on this analysis. The summarizer is implemented by a C++ program, createdigest, which reads in the raw directory listings and generates a list of MD5 digest values and URLs on a per-directory basis.

It might seem sensible to use all of the information from the directory listing output when creating the digest of it (i.e., permission, node, owner, group, size, date and the filename). Unfortunately most of these values can be changed during the mirroring process. In practice it seems that we can get by using only the file size and name (though as noted, some mirrors will compress files – we exclude these), plus the file type (e.g., plain file, directory or link).

We will note in passing that a better approach to persistence would be to calculate unique identifiers based on the contents of the files themselves, rather than the directory listing metadata. A program has been provided with the FTP Mirror Tracker distribution to let FTP server administrators calculate MD5 digest values for each file in a given directory hierarchy. However, this is very much a "future", since it would have to be widely deployed in order to be generally useful.

```
total 821
-r-xr-xr-x 1 root ftp  62163 Jan 25 19:43 compress
-r-xr-xr-x 1 root ftp 168240 Jan 25 19:38 date
-r-xr-xr-x 1 root ftp 106752 Jan 25 19:38 gzip
-r-xr-xr-x 1 root ftp 186848 Jan 25 19:37 ls
-r-xr-xr-x 1 root ftp 270232 Jan 25 19:38 tar
```

Listing 1: Typical output of the ls -lR command.

So, in order to produce the digest of a directory, we take the size and name of each of the files within it and concatenate them, e.g., in the example above we are left with:

```
62163compress168240date106752gzip186848ls270232tar
```

We then create a hexadecimal representation of the MD5 digest value for this string and assign it to the URL of the original directory on the FTP server. In this way we can create a relatively small unique identifier for the directory contents rather than for the files themselves. Finally, we add this text to the parent directory listing string, giving an identifier for the contents of the complete directory structure on the FTP server.

To understand why we need to add the unique identifier of the subdirectories to the string representation of the parent directory, let's take a simple example. Let's assume that someone wants to fetch the latest XFree86 for Linux runing on x86 with glibc 2.1 from the nearest server. This person starts to look for an exact replica nearby; see Table 1. If we hadn't

1	ftp://ftp.xfree86.org	no exact replicas
2	ftp://ftp.xfree86.org/pub/	no exact replicas
3	ftp://ftp.xfree86.org/pub/XFree86/	still no exact replicas
4	ftp://ftp.xfree86.org/pub/XFree86/4.0.1/	found some! so he shifts to the nearest one
5	ftp://ftp.gamma.ru/.3/XFree86/4.0.1/	
6	ftp://ftp.gamma.ru/.3/XFree86/4.0.1/binaries/	
7	ftp://ftp.gamma.ru/.3/XFree86/4.0.1/binaries/Linux-ix86-glibc21/	

Table 1: Search sequence for exactly replica.

added information about the subdirectories to the directory information, we couldn't be sure that the Linux-ix86-glibc21 directory is located somewhere on the mirror server under the 4.0.1 directory.

The Database Back End

We use a simple database design, with a dedicated database for the FTP Mirror Tracker data, and separate tables within this for each of the digest and link collections for each of the top level domains being tracked.

We are able to take advantage of the MySQL "load data" feature to read in the digest and link files directly, with MySQL automatically creating a database row in the appropriate table for each line of these files. The actual database manipulation is done using Perl, with DBI for database access.

The Digest Exchange

The original files are compressed and are moved to a directory which is accessible through the WWW, so that they can be shared with other Mirror Trackers.

Using the method described above we parse, for example, 100 MB of compressed listings from FTP servers (representing almost all of the anonymous FTP servers in Germany), create a 50 MB digest file and a 25 MB links file for feeding into the database engine. After compression, these files are 7 Mb and 2 Mb respectively – small enough to be shared easily via the WWW.

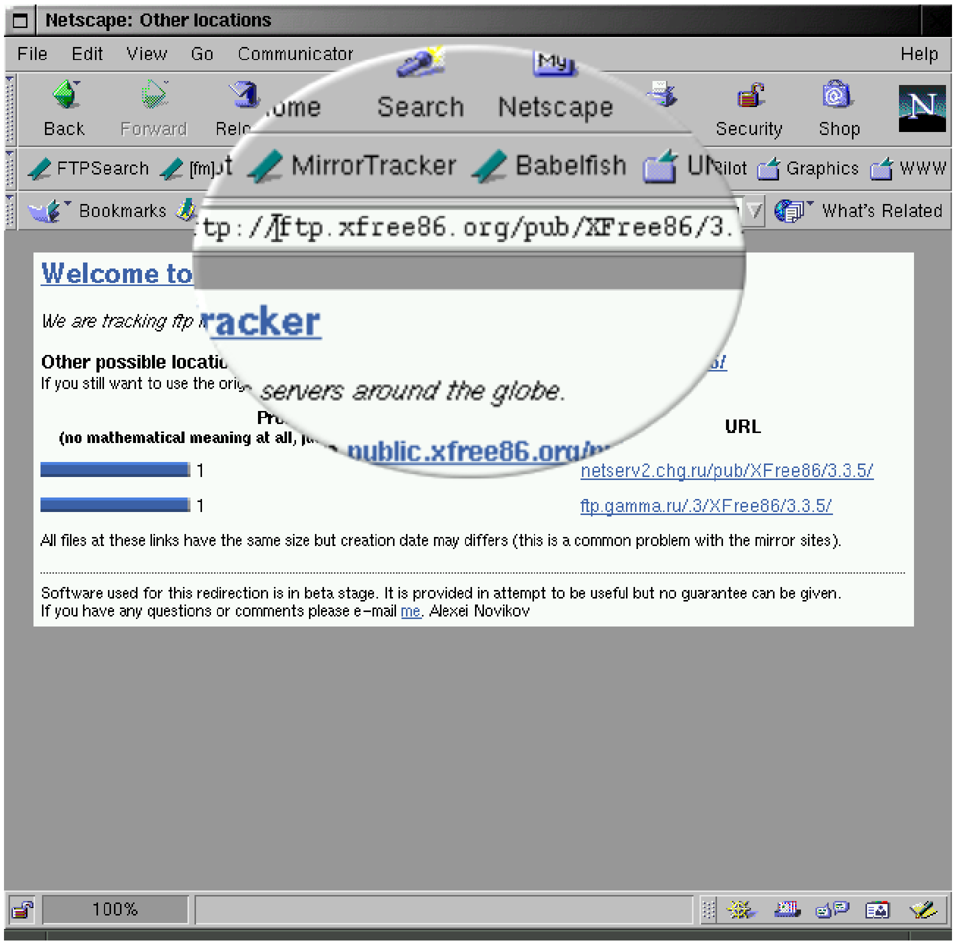
Frontend Programs

We provide a simple CGI program (written in Perl and C) which lets the end user query the FTP Mirror Tracker for specific URLs. This can be interacted with directly by the user, or linked to by content providers. There is also a third mode of operation which we will call Mirror Tracker on Demand.

Mirror Tracker on Demand is a JavaScript command which can be saved as a bookmark or (for example) placed on the Netscape Communicator Personal Toolbar (shown wrapped):

```
javascript:location.href='http://
    tracker.foo.bar//cgi-bin/
    tracker?url=' +
    escape(window.location);
```

Pressing this button will cause the current URL to be sent for resolution by the FTP Mirror Tracker, and the results of the Mirror Tracker search to be



tp://ftp.xfree86.org/pub/XFree86/3.

Welcome to

racker

We are tracking ftp

servers around the globe.

Other possible locatio

If you still want to use the orig

public.xfree86.org/

Pr

(no mathematical meaning at all,

URL

1 netserv2.chg.ru/pub/XFree86/3.3.5/

1 ftp.gamma.ru/.3/XFree86/3.3.5/

All files at these links have the same size but creation date may differs (this is a common problem with the mirror sites).

Software used for this redirection is in beta stage. It is provided in attempt to be useful but no guarantee can be given.
If you have any questions or comments please e-mail me. Alexei Novikov

Figure 1: Typical browser window with the output from the WWW frontend. "Mirror Tracker" button is zoomed.

presented in the browser's main window – as shown in Figure 1.

Although these user interfaces are very simple, they are also extremely powerful – instead of using search engines periodically to find out who is mirroring which sites, and trying to figure out which mirror sites are fresh and which are stale, content providers can have just one link on their site to the nearest FTP Mirror Tracker. The Mirror Tracker software will provide users with the list of the sites which have fresh and accurate replicas, optionally located in the same domain as the user.

Content providers only have to register with their nearest FTP Mirror Tracker server and put a link to it on their Web page. The link will look something like this (show here with lines wrapped for display purposes);

```
 <a href="http://squid.itep.ru/
   cgi-bin/tracker?url=YOUR URL">
   YOUR PACKAGE</a>
```

For example, the following link would yield all of the mirror sites known for the XFree86 3.3.6 release (shown wrapped):

```
 <a href="http://squid.itep.ru/
     cgi-bin/tracker?url=ftp://
     ftp.xfree86.org/pub/XFree86/
     3.3.6/">3.3.6 Version Mirrors</a>
```

Web Cache and Mirror Service Integration

Architecture for Cache/Mirror Cooperation

Whilst there is no standard network protocol for building and interacting with mirror services (rsync is probably the closest thing we have to this), there are a variety of ways in which Web Caching systems can be made to interoperate with each other – and in turn with Mirror services. In addition to the core proxy HTTP service, most Web Cache products also support interoperability via the Internet Cache Protocol [14], and the freeware Squid Web Cache [15] supports the new Cache Digest protocol [16]. Other cache cooperation protocols exist, such as the HyperText Caching Protocol [17], but these have yet to be as widely deployed and will not be further considered in this paper.

Since we do not wish to build our own Web Cache server, we have opted to use Squid for proxy HTTP services and graft on our own ICP server and Cache Digest generator – these are described in greater detail below.

Many Web Cache packages provide a URL rewriting facility, although some of these are limited to taking in a list of URLs to rewrite and the URLs to rewrite them to – or a list of URLs to block access to. So we can redirect desired URLs to the server with the FTP Mirror Tracker resolver.

Squid's approach, the "redirector" program, is particularly well suited to use with the FTP Mirror Tracker, since (when it is enabled) URLs are passed to an external program for analysis and potential rewriting. For performance reasons (the external program doing this may block whilst waiting for an operation to complete) Squid forks a large number of redirector programs which run continuously in parallel to the main Squid server process.

Use of the Internet Cache Protocol (ICP) for Cache Cooperation

The Internet Cache Protocol allows a Web Cache (or any other program which is interested in talking to Web Caches) to query another Web Cache as to its contents. Each ICP query or response takes the form of a UDP packet (with all fields in "network" or "big-endian" order), consisting of a twenty byte header with fields and a variable length payload. Typically the payload consists simply of a URL.

ICP is, for all its problems, the most widely implemented Web Cache cooperation protocol, available in most freeware and commercial caching products. Consequently, we decided to support it as an access method for the FTP Mirror Tracker too. Rather than modify Squid's own internal ICP server, which is very closely tied to the rest of Squid, a Perl module WebCache::ICP [18] was written to encapsulate the details of ICP packet processing within a simple object-oriented interface. This gives us an easy way to provide an ICP service which returns whatever ICP response we like on a given ICP request.

The ICP server we use with the FTP Mirror Tracker queries of the Mirror Tracker database to determine whether the Mirror Tracker should be visited for a given URL in the ICP request packet. In the case of downstream caches which are running Squid, we can reduce the amount of ICP traffic associated with the peering by using the "cache_peer_access" Access Control List, e.g., to specify that only FTP URLs should be sent to this peer.

Use of Cache Digests for Cache Cooperation

Cache Digests (implemented in Squid since version 1.2beta) offer a completely different approach to sharing information about cached objects, but have yet to be widely deployed in commercial Web Cache products. We decided to include them in the FTP Mirror Tracker project because of their widespread use with Squid, but the reader should note that they are still classified as experimental and subject to change.

Whereas ICP requires request and response UDP packets to be exchanged, Cache Digests work on the principle that the Web Cache builds a summary of its contents. This summary takes the form of a 128 byte header, followed by a bit array. This summary is then made available to other Web Caches via the normal proxy HTTP interface, and these are then in a position to do a local (Digest) lookup when trying to determine whether any of their peers has a requested URL.

In creating a Cache Digest populated from the Mirror Tracker database, we have the problem that there is no way to register whole URLs – the Mirror

Tracker database currently does not contain this information. Fuller Cache Digests could, however, be constructed usint the raw directory listing information which is processed in order to create the Mirror Tracker database.

Uniform Resource Names (URNs)

Introduction to URNs

URNs [19] are being developed by the Internet Engineering Task Force as an alternative naming scheme, complementary to the existing URL. Whereas URLs essentially encode a "recipe" of instructions to be followed when fetching a given copy of a resource, URNs are required to be:

- Location independent, so that the instructions for reaching the resource are de-coupled from the name by which the resource is cited, e.g., in Web documents.
- Persistent, so that the assignment of a URN to a resource effectively acts as a guarantee that the resource will continue to be accessible by this name indefinitely and will never be replaced by another resource which has the same URN.
- Compatible with existing name spaces such as ISSN and ISBN numbers.

In practice this is intended to be done by defining a number of "namespaces", each of which will have its own procedures for things like registration and management. URNs themselves [20] are simply the concatenation of the string "urn", a centrally assigned (by the Internet Corporation for Assigned Names and Numbers [21]) identifier for the namespace in question, and then a "Namespace Specific String" – separated by colons. That is:

urn:*Namespace Identifier*:Namespace Specific String

The authority responsible for each namespace [22] is free to subdivide the Namespace Specific String component of their URNs as they see fit – subject to character set encoding requirements which are designed to ensure that URNs may be used on the widest possible range of devices. The Namespace Identifier itself is chosen out of the set of upper and lower case letters, numbers and the hyphen character -, though the first character may not be a hyphen. The Namespace Specific String has a slightly larger vocabulary, which includes some punctuation characters.

URNs and the FTP Mirror Tracker

As noted above, the Mirror Tracker unique identifiers are subject to change over time. This means that they cannot be true URNs as these are defined by the IETF. However, since there is still a dearth of true URNs for people to experiment with, we shall ignore this limitation for the moment.

We supply a Squid compatible "N2L" [23] program with the FTP Mirror Tracker, using the Mirror Tracker "unique identifiers" as URNs and the experimental x-tracker Namespace Identifier. An example of a Mirror Tracker URN would be:

```
urn:x-tracker:57ce083433061aab97c9c2b63759ef2f
```

This is the unique identifier for all of the copies of the directory listing which can also be referred to by the URLs (as in the summarizer examples above):

```
ftp://ftp.xfree86.org/pub/XFree86/4.0/
ftp://netserv2.chg.ru/pub/XFree86/4.0/
ftp://ftp.gamma.ru/.3/XFree86/4.0/
ftp://caramba.cs.tu-berlin.de/pub/X/XFree86/4.0/
ftp://wizard.freesoftware.com/.0/XFree86/4.0/
ftp://ftp.linux.tucows.com/pub/XFree86/4.0/
```

Unfortunately, since even Netscape must be explicitly configured to talk to a proxy for URN resolution, we cannot simply deploy support for URNs as part of (for example) the JANET Web Cache Service and have them immediately be available to the end user. Each of the sites which connects to the service would need to enable URN support in their own users' browsers too.

Summary

The FTP Mirror Tracker enables transparent, user-controlled redirection to the nearest FTP mirror sites which are exact replicas of the original source. The redirection can be achieved by using a Web Cache server or by making an HTTP request to the FTP Mirror Tracker directly. The Mirror Tracker has internal URN support and can be used as a URN resolver for FTP requests.

During the course of this work we have produced a variety of tools and documents which may useful for other purposes above and beyond the FTP Mirror Tracker itself, e.g., the ICP and Cache Digest Perl modules, the Cache Digests specification, and the FTP robot. The software, databases, and documentation associated with the project are available for download via its homepage http://squid.itep.ru/. The changes which we made to Squid have been folded into the mainstream Squid 2.3 distribution, released in January 2000.

Acknowledgments

Discussions with Dave Beckett, Rodrigo Castro, Iain Fothergill, Claire Moore, George Neisser, Mark Russell and Michael Sparks are gratefully acknowledged.

The authors are grateful to the Trans-European Research and Education Networking Association for financial support under the Pilot Project initiative, and the Institute of Theoretical and Experimental Physics for hosting the FTP Mirror Tracker primary server and providing bandwidth for its operations.

Author Information

Martin Hamilton is a member of the Department of Computer Science, Loughborough University, UK & JANET Web Cache Service. Reach him electronically at martin@wwwcache.ja.net.

Since August 1997 Martin has been working on the JANET Web Cache Service, in the Computing

Services department of Loughborough University. The JANET Web Cache Service is funded centrally (with a contract awarded by open tender) for the use of the UK Education and Research community (users of the JANET network). With the instigation of usage-based charging for traffic from the US to JANET, this service has become extremely popular and has accounted for as much as half (46% on January 30th 2000) of the Web traffic transferred to JANET from the US. The JANET Web Cache Service is built upon open source software – the Linux and FreeBSD operating systems and the Squid Web Cache server.

In his spare time, Martin works as a volunteer on the GNU free software project at Massachusetts Institute of Technology and has contributed code to several popular open source products – including the NCSA and Apache World-Wide Web servers, the NCSA Mosaic WWW browser, and the Squid Web Cache.

Alexei Novikov has been a researcher at the Institute of Theoretical and Experimental Physics, Moscow, Russia. His email address is anovikov@heron.itep.ru since 1998. He defended his Ph.D. thesis (*Theoretical Restrictions on the Possible Extensions of the Standard Model Based on LEP Data*) in 1998. He works in the field of theoretical High Energy Physics (six papers in refereed journals, seven talks at the international conferences). He is interested in computer science and is developing several open source projects.

Bibliography

[1] *JANET website*, http://wwwcache.ja.net/ .

[2] *GNU wget homepage*, http://www.gnu.org/software/wget/ .

[3] *Mirror homepage*, http://sunsite.org.uk/packages/mirror/ .

[4] *rsync homepage*, http://rsync.samba.org/ .

[5] *UK Mirror Service*, http://www.mirror.ac.uk/ .

[6] *AARNet2 Mirror Archive*, http://www.aarnet.ebibitemrefu.au/projects/ .

[7] *Archie website*, http://archie.emnet.co.uk/ .

[8] *CNET shareware.com website*, http://shareware.cnet.com/ .

[9] *Fast FTP Search website,* FAST ASA, http://ftpsearch.lycos.com/ .

[10] R. Rivest, *RFC 1321, the MD5 Message-Digest Algorithm*, URN:ietf:rfc:1321, April 1992.

[11] *MySQL website*, http://www.mysql.com/ .

[12] *Perl website*, http://www.perl.org/ .

[13] *Perl DBI modules* at the Comprehensive Perl Archive Network, http://www.cpan.org/modules/by-module/DBI/ .

[14] Wessels, D. & K. Claffy *RFC 2186, Internet Cache Protocol (ICP), version 2*, URN:ietf:rfc:2186, September 1997.

[15] *Squid Web Proxy Cache Website*, http://www.squid-cache.org/ .

[16] Hamilton, M., A. Rousskov & D. Wessels, *Cache Digest Specification – Version 5,* http://www.squid-cache.org/CacheDigest/cache-digest-v5.txt , December 1998.

[17] Vixie, P., & D. Wessels, *HyperText Caching Protocol*, URN:ietf:rfc:2756, January 2000.

[18] M. Hamilton, *WebCache::ICP Perl Module* at CPAN, http://www.cpan.org/modules/by-module/WebCache/ .

[19] K. Sollins, *RFC 2276, Architectural Principles of Uniform Resource Name Resolution*, URN:ietf:rfc:2276, January 1998.

[20] Moats, R., *RFC 2141, URN Syntax*, URN:ietf:rfc:2141 May 1997

[21] *Internet Corporation for Assigned Names & Numbers website*, http://www.icann.org/ .

[22] Daigle, L., D. van Gulik, R. Iannella, P. Falstrom, *RFC 2611, URN Namespace Definition Mechanisms*, URN:ietf:rfc:2611, June 1999.

[23] Daniel, R., *RFC 2169, A Trivial Convention for using HTTP in URN Resolution*, URN:ietf:rfc:2169, June 1997.